

## Chapter 10

# Large Language Models as Optimizers

In this chapter, we present and discuss existing works that conceptualize LLMs as optimizers. First, we note that most existing studies focus on the prompt optimization problem defined in Equation (9.1), as optimizing other components of agentic workflows remains an emerging research area. To proceed, we draw parallels with classical iterative algorithms and examine their integration into modern optimization workflows.

### 10.1 Optimization Paradigms

Traditional optimization methods differ in their assumptions about objective function accessibility. We categorize them into three broad classes, each with an expanding level of input space: *gradient-based optimization*, which relies on explicit function gradients; *zeroth-order optimization*, which operates without gradient information; and *LLM-based optimization*, which extends beyond numerical functions to optimize over structured and high-dimensional input spaces.

- **Gradient-Based Optimization.** These methods assume access to gradient information and iteratively refine parameters. Techniques such as stochastic gradient descent (SGD) and Newton’s method [801] are widely used but require differentiability, limiting their applicability to discrete problems like prompt tuning and structured decision workflows, often endowed with a graph structure.
- **Zeroth-Order Optimization.** These methods bypass the need for explicit gradients by estimating search directions from function evaluations [802]. Examples include Bayesian optimization [803], evolutionary strategies [804], and finite-difference methods [805], which are effective when gradients are unavailable or expensive to compute. However, they still rely on well-defined numerical objectives and structured search spaces, which constrains their applicability to language-based tasks.
- **LLM-Based Optimization.** LLMs optimize broader solution spaces by leveraging natural language as both the optimization domain and feedback mechanism. By incorporating structured reasoning and human-like iteration, LLMs excel in refining prompts, generating adaptive workflows, and iteratively improving task performance based on user feedback.

While gradient-based and zeroth-order methods are typically applied to numerical objectives, their core principles, such as iterative refinement, search heuristics, and adaptive learning, also underlie LLM-based optimization strategies. Building on these insights, we highlight a rapidly emerging class of LLM-based optimization powered by reinforcement learning, which has become the backbone of slow thinking reasoning models [90, 806, 89]. As these models continue to evolve, we anticipate them driving the next wave of agentic applications, enabling LLMs to navigate complex environments with greater adaptability and strategic foresight.

### 10.2 Iterative Approaches to LLM Optimization

Some LLM-based optimization methods directly draw inspiration from classical optimization theory by adapting key components to address discrete and structured challenges. A central characteristic of these approaches is the iterative update step, in which model-generated modifications are selected from a range of possible improvements to refine the objective. Using the prompt optimization objective from Equation (9.1) as a running example, a general iterative

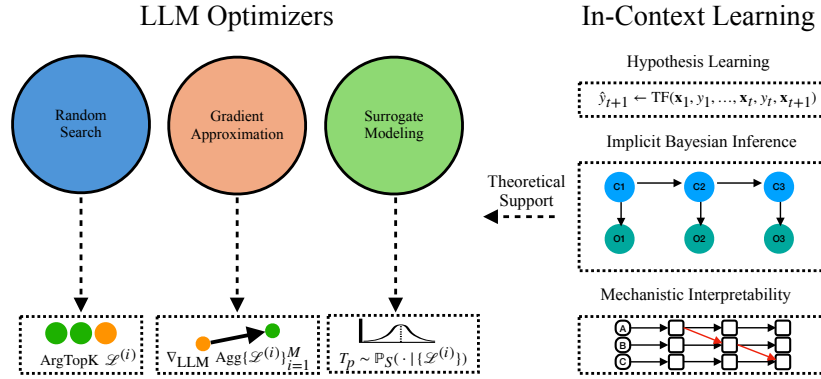


Figure 10.1: A taxonomy of LLM-based optimization methods, categorized into random search, gradient approximation, and surrogate modeling. We also highlight some theoretical explanations of in-context learning, which includes hypothesis learning, implicit Bayesian inference, and mechanistic interpretability, which underpin the optimization capabilities of LLMs.

algorithm can be expressed as follows:

**Sample:**  $T \sim \mathcal{D}$

**Evaluation:**  $\mathcal{L}(T; T_p) \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p), T)$

**Update:**  $T'_p \leftarrow \phi_{\text{opt}}(\mathcal{L}(T; T_p))$

Here, the **Sample** and **Update** steps are defined based on the agent’s task. In the simplest case, such as optimizing an instruction for binary classification of movie reviews, the objective  $\mathcal{L}$  is measured by classification accuracy. In more complex agentic workflows, the decision variable may include prompts at different workflow stages, tool selections, agent topologies, or a combination thereof. As discussed in Chapter 9, a common characteristic of these decision variables is their *combinatorial* nature—such as the set of all strings from an LLM’s vocabulary  $\mathcal{V}$  or all possible role assignments for agents in a workflow. Since enumerating all possible solutions is often intractable, this necessitates designing approximate update steps  $\phi_{\text{opt}}$ , which we discuss next.

- **Random Search.** Early LLM-based optimization methods leveraged random search variants to optimize prompts in discrete natural language spaces [774, 807, 651, 732, 808, 809, 810]. These methods often resemble evolutionary algorithms that iteratively sample candidate decision variables and select the top-performing ones from each iteration. The general formulation follows:

**Sample:**  $T \sim \mathcal{D}$

**Evaluation:**  $\mathcal{L}^{(i)} \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p^{(i)}), T), \quad i = 1, \dots, M$

**Update:**  $\{T_p^{(k)}\}_{k=1}^K \leftarrow \text{ArgTopK}_{i \in [M]} \mathcal{L}^{(i)},$

**Replenishment (Optional):**  $\{T_p^{(j)}\}_{j=K+1}^M \sim \text{Mutate}(\{T_p^{(k)}\}_{k=1}^K).$

We briefly override previous notations and let  $M$  denote the total number of candidate prompts sampled per iteration, and  $K$  (with  $K < M$ ) control the number of top-performing candidates—selected with ArgTopK in our algorithm—retained for the next step. This algorithm can optionally incorporate a replenishment step to maintain diversity in the candidate pool across iterations. Random search methods are simple to implement, highly parallelizable, and particularly effective for single-prompt workflows. Beyond prompt optimization, they have also demonstrated strong performance in selecting in-context demonstrations [811, 812]. However, their efficiency comes at a cost—each iteration requires  $O(M)$  parallel API queries, which can become prohibitively expensive for complex workflows involving multiple queries.

- **Gradient Approximations.** Several methods approximate gradient-based updates by iteratively refining solutions. For instance, [779, 730, 728] generate refinements at different workflow stages. StraGO [722] estimates descent directions using central-difference heuristics, while Trace [813] optimizes composed programs by modeling them as computation graphs, similar to backpropagation. The key analogy between gradient updates in continuous optimization and prompt-space refinement is the concept of a “descent direction”—a systematic *modification* of the decision variable to improve the objective. In contrast, random search methods propose new decision variables independently at each step, without accessing past update trajectories. Gradient-based approaches, by contrast, exploit this historical information, often leading to faster convergence. A general iteration for gradient approximation methods is given below:

$$\begin{aligned}
&\textbf{Sample: } T^{(i)} \sim \mathcal{D}, \quad i = 1, \dots, M \\
&\textbf{Evaluation: } \mathcal{L}^{(i)} \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p), T^{(i)}), \quad i = 1, \dots, M \\
&\textbf{Gradient Approximation: } g \leftarrow \nabla_{\text{LLM}} \text{Agg}(\mathcal{L}^{(1)}, \dots, \mathcal{L}^{(M)}) \\
&\textbf{Update: } T'_p \leftarrow \phi_{\text{opt}}(T_p, g),
\end{aligned}$$

where  $M$  is the minibatch size,  $\text{Agg}(\cdot)$  is an aggregation function that combines feedback signals (e.g., in numerical optimization,  $\text{Agg}$  is typically the average operator),  $\nabla_{\text{LLM}}$  represents an abstract “LLM-gradient operator” [728] that generates textual refinement directions based on the feedback signal and the current minibatch (e.g., *the agent should consider the edge case of ...*). Additionally,  $\phi_{\text{opt}}$  can be instantiated as an LLM query, allowing the agent to update its prompt based on  $g$ .

Compared to random search methods, gradient-based approaches offer two key advantages: they enable the incorporation of past refinement directions into  $\phi_{\text{opt}}$ , analogous to momentum-based techniques in first-order optimization algorithms [814, 815], and they facilitate backpropagation-like techniques for optimizing computation graphs [651, 813, 780], making them particularly effective for multi-stage workflows with interdependent optimizable modules. However, this flexibility comes at the cost of increased design overhead, such as the need for meta-prompts to aggregate feedback and apply refinement directions. We further discuss the feasibility of using LLMs to optimize these *hyperparameters* below. Some approaches also explored direct gradient-based optimization of soft prompts [816, 817, 818]. While effective for simple input-output sequence learning, these methods struggle with multi-step workflows and sequential decision-making [630, 300].

Finally, while these methods leverage first-order optimization insights, the extension of second-order techniques (e.g., quasi-Newton methods) to LLM-based optimization remains largely unexplored. Fortunately, recent works such as Revolve [780] have taken a step in this direction by introducing a structured approach for second-order optimization, modeling the evolution of response patterns over multiple iterations. By incorporating higher-order refinements, Revolve enables more stable and informed optimization, effectively mitigating stagnation in complex tasks. We are also excited by emerging trends in leveraging inference-time compute [90, 89] to incorporate historical refinement directions and investigate the benefits of momentum.

- **Bayesian Optimization and Surrogate Modeling.** While the aforementioned approaches achieved significant progress in LLM-based optimization, they often entail substantial financial and environmental costs due to the high number of required LLM interactions. Moreover, these methods can be sensitive to noise, and the optimization landscape of discrete prompts, among other decision variables, remains poorly understood [819, 820]. Under these constraints, Bayesian Optimization (BO) emerges as a compelling alternative, as it builds a noise-resilient surrogate model of the optimization objective:

$$\begin{aligned}
&\textbf{Sample: } T \sim \mathcal{D} \\
&\textbf{Proposal: } \{T_p^{(i)}\}_{i=1}^M \sim S. \text{Propose} \\
&\textbf{Evaluation: } \mathcal{L}^{(i)} \leftarrow \phi_{\text{eval}}(\phi_{\text{exe}}(Q, T_p^{(i)}), T), \quad i = 1, \dots, M \\
&\textbf{Update: } S \leftarrow S. \text{UpdatePrior}(\{\mathcal{L}^{(i)}\}_{i=1}^M, \{T_p^{(i)}\}_{i=1}^M),
\end{aligned}$$

where  $S$  represents a probabilistic surrogate model of the optimization objective, equipped with a proposal operator (e.g., posterior sampling from a Gaussian Process BO procedure [803]) and an update mechanism based on observed evidence from prompt evaluations. For instance, MIPRO [821] employs a Tree-Structured Parzen Estimator as its surrogate [822], while PROMST [823] trains a score-prediction model to guide prompt tuning. Leveraging a surrogate model for LLM-based optimization aligns with the emerging trend of amortized optimization for non-differentiable objectives [824]. For instance, [825] trains a prompt-generator LLM to amortize the computational cost of instantiating a beam search problem for discovering jailbreak attack prefixes.

Finally, several other works fit an additional lightweight module—such as a Bayesian belief posterior or a utility function—from LLM outputs, to aid the optimization of domain-specific workflows, such as decision-making and multi-agent negotiations [826, 827]. This type of amortized methods—those that fit a parameterized model that is reusable for unseen inputs—have found increasing usage in LLM-based optimization, such as jailbreaking [828, 825].

### 10.3 Optimization Hyperparameters

Similar to traditional optimization, LLM-based methods are highly sensitive to hyperparameters that influence search efficiency and generalization. A key consideration in gradient-based LLM optimizers is the choice of the aggregation function  $\text{Agg}(\cdot)$ , which determines how textual feedback is synthesized to guide prompt updates. An improper choice can lead to loss of critical information or misalignment in iterative refinements. Additionally, [813] introduces a “whiteboard” approach, where an LLM program is decomposed into human-interpretable modules. However, design choices in structuring such modular workflows remain largely unexplored, which poses an open challenge for optimizing LLM-driven decision-making pipelines.

Hyperparameters in LLM optimization often parallel those in numerical optimization. For example, batch size plays a crucial role: just as minibatch updates enhance stability and efficiency in classical optimization, LLM-based approaches like TextGrad [728] aggregate feedback across multiple generated samples before making updates. Another key factor is momentum—while it stabilizes updates in gradient-based methods by incorporating past gradients, LLM-based optimizers similarly leverage historical refinements to improve performance over time [728, 813]. Despite progress in numerical optimization, hyperparameter selection for LLM-based optimizers remains largely heuristic, often relying on ad hoc, trial-and-error tuning.

In agentic system design, hyperparameters proliferate across various components, including role assignments of agents, selection of in-context demonstrations, and scheduling of tool invocations. Each of these choices has a profound impact on downstream performance, yet principled methods for optimizing them remain underdeveloped. While traditional hyperparameter tuning techniques, such as grid search and Bayesian optimization, can be applied to discrete LLM-driven workflows, their computational cost scales poorly due to the high variance in language model outputs. Additionally, the combinatorial nature of these hyperparameters, where agent configurations, prompting strategies, and reasoning structures interact in complex ways, makes an exhaustive search infeasible. Recent work has attempted to address this challenge by embedding agentic workflows into structured frameworks such as finite state machines [729], optimal decision theory [826], and game theory [827]. However, these approaches often fail to generalize across diverse environments. A promising direction for addressing these challenges is meta-optimization, where LLMs are used to optimize their own hyperparameters and decision-making strategies. For example, an LLM-based optimizer can iteratively refine its own prompting strategies by treating past decisions as experience, akin to learned optimizers in deep learning [829]. Moreover, amortized approaches train auxiliary models to predict effective hyperparameters, which can reduce the computational cost of exhaustive search [821, 823]. While these techniques offer exciting possibilities, they also introduce new challenges, such as balancing exploration with exploitation in adaptive tuning and ensuring generalization across diverse optimization tasks. Investigating principled meta-optimization strategies tailored to LLM-driven workflows remains a critical area for future research.

### 10.4 Optimization across Depth and Time

Unlike conventional optimizers that update parameters in a static setting, LLMs optimize workflows dynamically, considering both depth (single-pass workflows) and time (recurrent updates). In terms of depth, LLMs function similarly to feedforward networks, sequentially optimizing workflows as they pass through different modules—most existing LLM-based optimizers follow this paradigm. Beyond single-pass execution, LLMs can also optimize over time, akin to recurrent architectures such as RNNs or Universal Transformers [830], by iteratively refining decision-making. For instance, StateFlow [729] enhances workflows by incorporating feedback across multiple iterations, enabling dynamic refinement and adaptation over time. While these analogies are compelling, many well-established engineering optimization techniques—such as checkpointing [831] and truncated backpropagation [832]—remain underexplored in LLM-based optimization. We see this as a promising avenue for future research, echoing previous calls for deeper investigation [813].

## 10.5 A Theoretical Perspective

Recent studies suggest that transformers inherently perform optimization-like computations, supporting their potential as general-purpose optimizers for computational workflows. However, a significant gap remains between their empirical success and theoretical understanding. Here, we provide a brief overview of recent progress in bridging this gap.

- **In-Context Learning.** A fundamental perspective on transformers as optimizers emerges from in-context learning, particularly in few-shot settings [2]. [733] demonstrated that transformers can in-context learn diverse regression hypotheses, including regularized linear models, decision trees, and shallow neural networks. Building on this, later works [734, 833, 735] provided constructive proofs that transformers can implement iterative optimization algorithms, such as gradient descent and second-order updates. However, while these theoretical models characterize transformers’ optimization capabilities, they do not fully explain in-context learning in large-scale LLMs, which operate in discrete input-output spaces. Empirical analyses [819, 834, 820] instead sought to understand how pre-trained LLMs generalize in-context. [834] proposed that in-context learning resembles a hidden Markov model (HMM) performing implicit Bayesian inference, while [819, 820] challenged the conventional view that in-context demonstrations serve as new test-time samples for hypothesis formation. In-context learning remains the central emergent ability [835] enabling self-improvement and optimization from context, yet it continues to elude comprehensive theoretical analysis.
- **Mechanistic Interpretability.** Parallel to theoretical analyses, mechanistic interpretability aims to uncover internal transformer computations by identifying subgraphs, also known as circuits, responsible for specific behaviors. Early studies mapped circuits for stylized language tasks in pre-trained GPT-2 models [836, 837, 838], while more recent efforts have scaled up by identifying semantically meaningful features using sparse autoencoders [839, 736, 840, 841]. These methods have been largely successful in eliciting causal and controllable behavior from frontier-class LLMs, but they also reveal an unintended consequence: in-context learning capabilities often entangle beneficial generalization with harmful behaviors when conditioned on many-shot demonstrations [842]. This raises challenges for optimizing LLM workflows safely and reliably.
- **Limitations Under Uncertainty.** While LLMs demonstrate moderate capabilities in sequential decision-making when provided with in-context information, they struggle to make optimal choices under uncertainty [843, 844, 845, 846]. In particular, [826] found that LLM-based optimizers exhibit difficulty in adapting to stochastic environments, often failing to explore optimally. These findings serve as a cautionary note for deploying LLM-based optimizers in dynamic or uncertain settings where exploration and robust decision-making are critical.

LLMs redefine optimization by integrating structured reasoning, natural language processing, and in-context learning, expanding beyond traditional numerical methods. Despite strong empirical performance in structured search spaces, open questions remain about the theoretical underpinnings of LLM-based optimization, particularly the emergence of in-context learning from standard gradient-based training.